CORRECTED VERSION*

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 15/00

A1

(11) International Publication Number: WO 98/02822

(43) International Publication Date: 22 January 1998 (22.01.98)

(21) International Application Number: PCT/US97/12516

(22) International Filing Date: 17 July 1997 (17.07.97)

(30) Priority Data:
08/683,625 17 July 1996 (17.07.96) US
60/036,526 29 January 1997 (29.01.97) US

(60) Parent Applications or Grants
(63) Related by Continuation
US 08/683,625 (CON)
Filed on 17 July 1996 (17.07.96)
US 60/036,526 (CON)
Filed on 28 January 1997 (28.01.97)

(71) Applicant *(for all designated States except US)*: I.Q. SYSTEMS, INC. [US/US]; 75 Glen Road, Sandy Hook, CT 06482 (US).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: ROBINSON, Jeffrey, I. [GB/US]; 37 Eastview Drive, New Fairfield, CT 06812 (US).

(74) Agent: GORDON, David, P.; 65 Woods End Road, Stamford, CT 06905 (US).

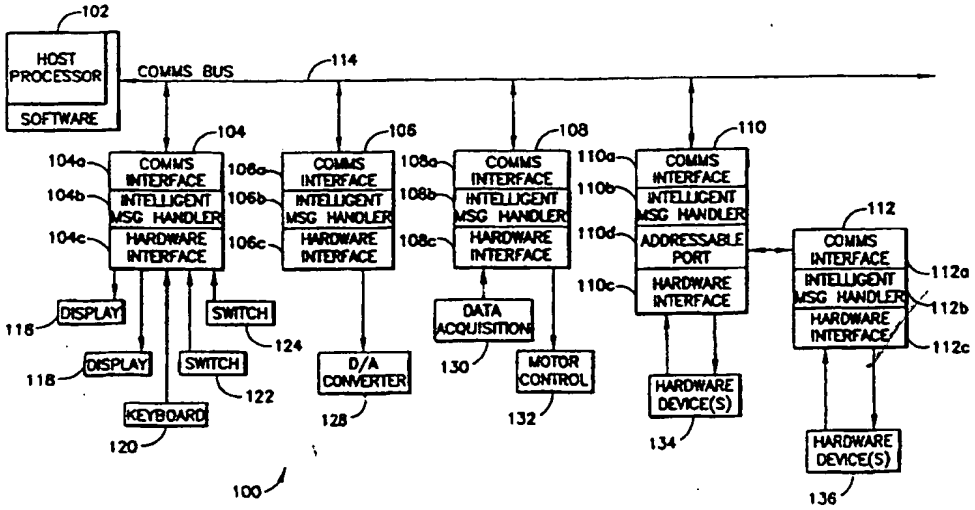
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, US, UZ, VN, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHODS AND APPARATUS FOR DISTRIBUTED PROCESSING UTILIZING OBJECT ORIENTED PROCESSORS AND FOR RAPID ASIC DEVELOPMENT

(57) Abstract

The invention provides a distributed processing system (100) having a host processor (102) and one or more object oriented processors (104) which are embodied as discrete components and as a collection of components on a single ASIC chip. A high level command language and a communications bus system are also provided both for use with discrete components and as an integral part of an ASIC chip. The ASIC chips are premanufactured to operate identically to a corresponding collection of discrete components. A distributed processing system (100) is developed by coupling a collection of discrete object oriented processors (104) and a host processor (102) to a bus (114) and writing a command language script to define the functionality of the system. After the system is designed and tested using discrete components, a suitable premanufactured ASIC or collection of ASICs is chosen and coupled to a host processor (102). The high level command language script permits the host processor (102) and the ASIC system to perform identically to the discrete component system. An addressing scheme is provided in the command language such that multiple object oriented processors (104) may be given a single simultaneous command. Object oriented processors (104) may have child ports with soft addressability to which other object oriented processors (104) may be coupled. The child ports wich soft addressability have software assignable addresses.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 15/00

A1

(11) International Publication Number: WO 98/02822

(43) International Publication Date: 22 January 1998 (22.01.98)

(21) International Application Number: PCT/US97/12516

(22) International Filing Date: 17 July 1997 (17.07.97)

(30) Priority Data:

08/683,625	17 July 1996 (17.07.96)	US
60/036,526	29 January 1997 (29.01.97)	US

(71) Applicant *(for all designated States except US)*: I.Q. SYSTEMS, INC. [US/US]; 75 Glen Road, Sandy Hook, CT 06482 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: ROBINSON, Jeffrey, I. [GB/US]; 37 Eastview Drive, New Fairfield, CT 06812 (US).

(74) Agent: GORDON, David, P.; 65 Woods End Road, Stamford, CT 06905 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, US, UZ, VN, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published

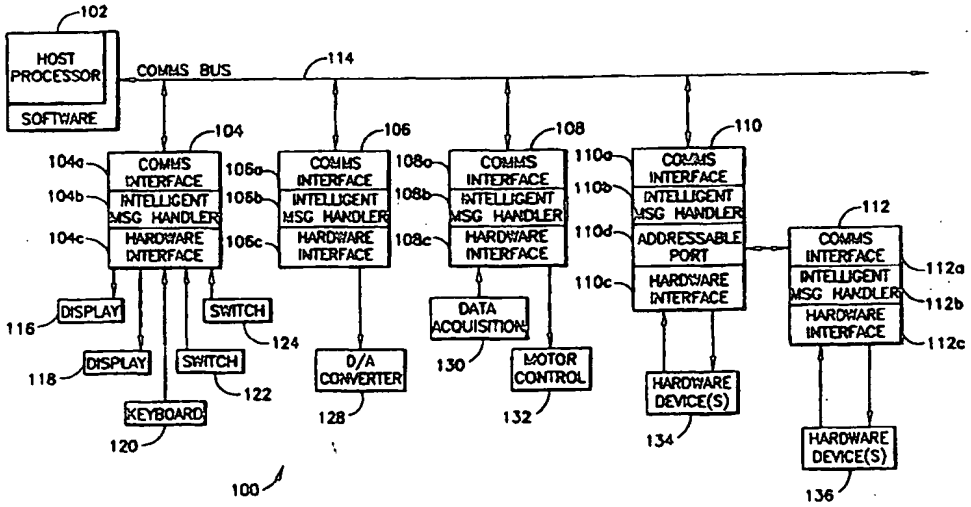
With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHODS AND APPARATUS FOR DISTRIBUTED PROCESSING UTILIZING OBJECT ORIENTED PROCESSORS AND FOR RAPID ASIC DEVELOPMENT

(57) Abstract

The invention provides a distributed processing system (100) having a host processor (102) and one or more object oriented processors (104) which are embodied as discrete components and as a collection of components on a single ASIC chip. A high level command language and a communications bus system are also provided both for use with discrete components and as an integral part of an ASIC chip. The ASIC chips are premanufactured to operate identically to a corresponding collection of discrete components. A distributed processing system (100) is developed by coupling a collection of discrete object oriented processors (104) and a host processor (102) to a bus (114) and writing a command language script to define the functionality of the system. After the system is designed and tested using discrete components, a suitable premanufactured ASIC or collection of ASICs is chosen and coupled to a host processor (102). The high level command language script permits the host processor (102) and the ASIC system to perform identically to the discrete component system. An addressing scheme is provided in the command language such that multiple object oriented processors (104) may be given a single simultaneous command. Object oriented processors (104) may have child ports with soft addressability to which other object oriented processors (104) may be coupled. The child ports wich soft addressability have software assignable addresses.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

METHODS AND APPARATUS FOR DISTRIBUTED PROCESSING UTILIZING OBJECT ORIENTED PROCESSORS AND FOR RAPID ASIC DEVELOPMENT

This application claims priority benefit from Serial Number 08/683,625, filed July 17, 1996 for "Method And Apparatus For Distributed Processing And Rapid ASIC Development", and provisional application Serial Number 60/036,526 filed January 28, 1997 for "Object Oriented Processors with Enhanced Application Directed Flexibility".

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to distributed processing systems. More particularly, the invention relates to a distributed processing system wherein discrete components communicate via a high level language such that functional systems may be rapidly developed and reduced to an ASIC implementation without the timing and electrical problems usually associated with ASIC development.

2. State of the Art

Application Specific Integrated Circuits (ASICs) are widely used to implement sophisticated circuits for mass production. ASICs are developed in a number of different ways. One approach is to construct a prototype using discrete components wired together on a bread board or an etched circuit board, test and debug the prototype, and then migrate the circuit to an ASIC implementation. This approach has the advantage that the concept of the circuit is proven prior to ASIC implementation. A significant disadvantage of this approach is that the ASIC implementation utilizes very low level building blocks (e.g., individual gates and registers) on a single chip which perform differently than the discrete components used in the prototype. The differences in performance of the on-chip components manifests itself in timing differences, parasitic capacitance, power balancing requirements, etc. Moreover, given the

differences between the ASIC components and the prototype components, the mapping of the prototype to an ASIC chip may not be an optimal or even feasible implementation. Substantial redesign may be required during the migration to ASIC and the redesign introduces additional risk that the circuit will not perform as desired.

Another popular approach to ASIC design is to simulate a circuit using computer software which relies on a cell library supported by the ASIC manufacturer. Although this approach also involves careful attention to timing and the electrical characteristics of the final chip, there is greater confidence that the final chip will perform in the same manner as the simulated circuit. Nevertheless, software simulation is not always an accurate substitute for a real world prototype. The concept of the circuit is not tested in real world conditions until after the ASIC prototype is delivered, i.e. after a significant investment of time and money. Although the ASIC may perform according to the specifications of the software simulation, it may not be completely functional if the specifications are incorrect for a real world application. The development of an accurate specification for a real world circuit without any hardware testing is difficult and time consuming.

In most of the approaches to ASIC design and manufacture, once the circuit is reduced to an ASIC chip it cannot be easily and efficiently modified without redesigning a new chip. While there do exist field programmable gate arrays (FPGAs), these devices are too inefficient, because of their large die size, for higher volume applications. ASICs sacrifice flexibility for the ability to perform a specific set of functions in an efficient low cost manner.

My previous application U.S. Serial Number 08/525,948, which is hereby incorporated by reference herein, discloses distributed processing systems having a host processor and at least one object oriented processor. An object oriented processor has a communications interface, an intelligent message handler, and a

task-specific functionality. The task-specific functionality is somewhat configurable and an application developer need not utilize all of the functionality of each object oriented processor. In some situations, it may be desirable to provide multiple iterations of a specific functionality which can result in the underutilization of several object oriented processors.

Each object oriented processor has a functionality which defines its physical connectability. More specifically, as embodied on a single chip, each object oriented processor presents a number of pins for coupling the processor to other devices. According to present embodiments of the object oriented processors, the functionality of each pin is substantially fixed at the time the object oriented processor is manufactured. For example, as disclosed in my previous application Serial Number 08/525,948, a user interface controller utilizes thirty-seven pins, most of which have a set functionality. Several of the pins have alternate functionality. For example, pins A0 through A7 are an aux port. However, pins A1 and A2 can be used as LCD enable pins and pins A3-A7 can be used as LED enable pins. Nevertheless, for the most part, the functional resources of the object oriented processors are pre-defined with respect to certain pins and cannot be substantially changed by the developer/user.

SUMMARY OF THE INVENTION

It is therefore an object of the invention to provide methods and apparatus for rapid ASIC development.

It is also an object of the invention to provide methods and apparatus for rapid ASIC development which avoids the timing and electrical problems typically associated with ASIC development.

It is another object of the invention to provide methods and apparatus for distributed processing which includes tools for rapid reconfiguration of a distributed processing system without requiring any hardware modification.

It is still another object of the invention to provide methods and apparatus for rapid ASIC development wherein the functionality of the ASIC chip can be easily and efficiently reconfigured after fabrication through the use of software commands.

It is another an object of the invention to provide an object oriented processor with enhanced post-manufacture configurability.

It is also an object of the invention to provide an object oriented processor on a chip having a number of pins where the functionality of the pins is configurable by a developer/user.

It is another object of the invention to provide an object oriented processor which contains a library of functionality which may be selected in virtually any combination.

It is still another object of the invention to provide an object oriented processor which contains a library of functionality which may be which may be assigned to pins via software commands.

In accord with these objects which will be discussed in detail below, one embodiment of the present invention draws on aspects of the parent applications to provide methods and apparatus for ASIC design. In addition, the invention provides enhancements to the technology of the parent applications which allows for rapid reconfiguration of a distributed processing system without requiring any hardware modification. Accordingly, the invention provides methods and apparatus for "post-fabrication optimization" of ASIC chips.

The apparatus of the invention therefore provides a distributed processing system having a host processor and one or more object oriented processors which are embodied as discrete components and as a collection of components on a single ASIC chip. A high level command language and a communications bus

system are also provided both for use with discrete components and as an integral part of an ASIC chip. The ASIC chips according to the invention are premanufactured to operate identically to a corresponding collection of discrete components. According to the methods of the invention, a distributed processing system is developed by coupling a collection of discrete object oriented processors and a host processor to a bus according to the invention and writing a command language script to define the functionality of the system. After the system is designed and tested using discrete components, a suitable premanufactured ASIC or collection of ASICs is chosen and coupled to a host processor using the same bus structure as the discrete component system. The high level command language script permit the host processor and the ASIC system to perform identically to the discrete component system.

The methods of the invention also provide an addressing scheme in the high level command language such that multiple object orient processors may be given a single simultaneous command. In addition, the apparatus of the invention provides "parent" object oriented processors which have one or more child ports with soft addressability to which other "child" object oriented processors may be coupled. The child ports with soft addressability have software assignable addresses and the address of an addressable port may be changed using the high level command language. The "virtual address" of each child processor can therefore be changed at any time without any change in the hardware of the distributed processing system. The methods of the invention further include device interrogation wherein the host determines the functionality of each of the object oriented processors via a command, functional grouping of the object oriented processors by the host through the assignment of virtual addresses, and reconfiguration of the parameters of object oriented processors by the host via commands.

Utilizing the methods and apparatus of the present invention, distributed processing systems can be constructed using discrete components, the functionality of the system

defined using the high level command language, the topology and functionality of the system altered through the use of virtual addressing, and the entire system replicated on one or more ASIC chips which function identically to the discrete component system. In addition, the functionality of the system as embodied on one or more ASIC chips may be further altered using the high level command language without the need to modify any of the hardware.

According to a presently preferred embodiment of the invention, an object oriented processor includes a readable memory containing a library of configurable (programmable) functions and a writable memory in which functions are instantiated and configured. More specifically, the object oriented processor includes a system functionality which is automatically instantiated in writable memory at power-up, which maintains an active task list, and which calls other functions to be instantiated in writable memory in response to commands from a host processor. The object oriented processor according to the invention further includes a communications interface, a global input parser, an output message former, an output message registration function, and a scheduler for the purpose of communicating with a host or other processor. A host processor is used to configure the object oriented processor utilizing a high level command language and may also be used to communicate with the object oriented processor during subsequent post-configuration operations. Optionally, a boot ROM may be used to automatically configure the object oriented processor at power-up.

According to the presently preferred embodiment of the object oriented processor, the library of functions is configured as a library of cells (objects) stored in ROM. Each cell (object) includes a functional layer, a parser layer, a timing layer, and an instantiation layer. The system functionality, also referred to as the system object, is also stored in ROM and is automatically instantiated when the processor is powered on and reserves RAM for the active task list (pointers to

instantiated cells). The system object is similar to a cell but handles global methods and functions which are common to all cells. A memory manager is provided for allocating RAM to instantiated cells. In response to a high level command from a host processor, the system object calls the instantiation layer of a cell in the cell library and commands the cell to instantiate itself in RAM. The instantiation layer of the cell calls the memory manager and requests an allocation of RAM. The memory manager returns a pointer (to a starting address in RAM) to the cell which the cell stores in a pre-reserved portion of RAM. Meanwhile, the system object stores a pointer to the cell in the active task list. According to the presently preferred embodiment, the location of the pointer in the task list is used as the "instantiation name" for this instantiation of the cell. The cell arranges its allocated RAM in two parts. The first part is the output message header which includes a pointer to the output buffer of the cell instantiation, the interrupt priority of the cell instantiation, the message type, the source ID, the instantiation name, and the destination cell and processor. The second part is private data used by the cell in the performance of its functionality. Once a cell has been thus instantiated, pins can be assigned to the instantiation of a cell by sending a high level command from a host processor to the object oriented processor. The global input parser checks the syntax of all incoming messages, buffers the message, examines the command and looks at the active task list to determine the cell to which the command is directed. The parser layer of the cell looks at the instantiation name of the cell instantiation to which the command is addressed and reads its preassigned reserved area of RAM to find the instantiation of the cell bearing that name. The parser layer also interprets the pin assignment message and stores the pin assignment in its private data area of RAM for the named instantiation of the cell.

According to other preferred aspects of the invention, cells may be instantiated several times so long as enough hardware resources (pins and RAM) are available to support another instantiation. Each cell keeps track of its own instantiations

by writing to a pre-assigned reserved area of memory as a static variable. Prior to instantiating itself, a cell looks to its reserved area of memory to determine how many instantiations of itself there are and whether there exist enough hardware resources for a further instantiation. In addition, the memory manager maintains a pointer to the memory heap which is utilized and generates an error message if requested to assign more RAM than is available. After pins have been assigned to an instantiation of a cell, an interrupt priority can be assigned. According to the presently preferred embodiment, an interrupt priority of 0-7 may be assigned where 0 represents polled priority and 1-7 indicate increasing levels of importance. When the priority level is greater than zero, output messages will be generated autonomously. The interrupt priority is stored by the instantiation of a cell in its output message header part of RAM. During setup, a main task dispatcher in the system object initiates a global variable indicating the number of active tasks =0. Each time a cell is instantiated, this variable is incremented. When the variable is >0, the system object repeatedly scans the active task list. Each time a cell is instantiated, all active tasks are stopped and all instantiated cells are called to their timing layer and the tasks are scheduled. The system object assigns an offset to each cell instantiation which the timing layer stores in private data. The cell returns a worst case time to the system object and the worst case time is used to calculate the offset for the next active task. The time between the worst case and the actual time is advantageously used by the system object for system (background) functions; i.e. system functions are not otherwise scheduled and therefore do not require overhead.

During operation, a message to a particular instantiation of a cell is parsed by the global input parser which consults the active task list to find the pointer to the cell. The cell receives the message from the global input parser and scans through its reserved memory to find which instantiation of itself is being addressed. Messages from an instantiation of a cell are placed in the output message header part of its RAM and a request

for registration is sent to the output registrar. The registrar maintains a list of pointers to message headers (which contain priority values) and raises a flag for the cell when an outgoing message has been queued. So long as the flag is raised, the cell can not receive input. The output message former scans the pointers in the list examining priority takes the information from the output message headers to create messages for the host processor or other processor on the network. The output message former sends the message and drops the flag after the message has been sent.

According to other preferred aspects of the invention, a high level language is provided for communication between object oriented processors and a host processor during set-up and during operation.

The functionality of each cell in the cell library may be chosen from a broad range of functions such as those discussed in the parent applications. Object oriented processors according to the invention may be provided with a set of complementary functionalities or may be provided with a broad range of different functionalities.

Additional objects and advantages of the invention will become apparent to those skilled in the art upon reference to the detailed description taken in conjunction with the provided figures.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a schematic block diagram of a distributed processing system according to the invention in which peripheral devices are coupled to a host processor via a communications channel and discrete object oriented processors;

Figure 2 is a schematic block diagram of a distributed processing system according to the invention in which peripheral

devices are coupled to a host processor via a communications channel and object oriented processors embodied on an ASIC chip;

Figure 3 is a diagram similar to Figure 2 in which the host processor is also embodied on the same ASIC chip as the object oriented processors;

Figure 4 is a schematic block diagram of a "parent" object oriented processor according to the invention with the enhanced capability of an addressable port;

Figure 5 is a functional block diagram of a preferred object oriented processor according to the invention;

Figure 6 is a functional block diagram of a predefined functional cell according to the invention;

Figure 7 is a schematic block diagram of an object oriented processor according to the invention coupled to a host processor and a power supply;

Figure 8 is a schematic memory map of the writable memory area in the object oriented processor of Figure 5;

Figure 9 is a schematic flow chart illustrating steps in the setup programming of the preferred object oriented processor according to the invention; and

Figure 10 is a schematic flow chart illustrating the operational mode of the preferred object oriented processor according to the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring now to Figure 1, an exemplary development system 100 according to the invention includes a host processor 102, and a plurality of object oriented processors 104, 106, 108, 110 which are coupled to the processor 102 via a communication bus

114. Preferably, the bus 114 utilizes one of the "zero-dominant" protocols described in my previous application U.S. Serial Number 08/545,881 filed October 20, 1995, or my previous application U.S. Serial Number 08/645,262 filed May 13, 1996, both of which are hereby incorporated by reference herein in their entireties.

Each of the object oriented processors 104, 106, 108, 110 includes a Comms interface 104a, 106a, 108a, 110a an intelligent message handler 104b, 106b, 108b, 110b and a functional layer which in the examples shown is a hardware (peripheral) interface 104c, 106c, 108c 110c. In accord with the invention, however, the functional layers need not be hardware related, but could be a processing routine. Each object oriented processor 104, 106, 108, 110 is bidirectionally coupled via its respective Comms interface 104a, 106a, 108a, 110a to the communication bus 114 which is coupled to the host processor 102. This system is substantially the same as the system described in my previous application U.S. Serial Number 08/525,948 except for the object oriented processor 110 which includes an addressable port 110d through which another object oriented processor 112 is coupled. The object orient processor 112 is similar to the object oriented processors 104, 106, 108 and is provided with a Comms interface 112a, an intelligent message handler 112b, and a functional layer 112c. As described in more detail below, a high level command language is provided for communication among the host and the object oriented processors 104, 106, 108, 110, 112.

As shown in Figure 1, each of the exemplary object oriented processors 104, 106, 108 is designed to support different types of peripherals.

The first object oriented processor 104 (which is described in detail in the parent application) has the task of a universal front panel controller (a user interface controller). It is designed to support peripheral devices such as LED/LCD alphanumeric displays 116, 118, a keypad or keyboard 120 (which is actually a matrix of switches), and several rotary encoders or switches 122, 124. As such, the hardware interface 104c is

specifically designed to accommodate these peripheral devices. Moreover, the intelligent message handler 104b of this object oriented processor 104 need only respond to messages appropriate for the types of peripherals serviced by it.

The second object oriented processor 106 (which is described in detail in my previous application application) is a speech messaging controller and is designed to support a digital-to-analog converter 128. Consequently, the hardware interface 106c is specifically designed to accommodate these peripheral devices. Moreover, the intelligent message handler 106b of this object oriented processor 106 need only respond to messages appropriate for the types of peripherals serviced by it.

The third object oriented processor 108 (which is described in detail in the parent application) has the task of an analog interface and is designed to support analog data acquisition devices 130 and pulse width modulation controlled analog devices such as power supplies and motor controls 132. As such, the hardware interface 108c is specifically designed to accommodate these peripheral devices. Moreover, the intelligent message handler 108b of this object oriented processor 108 need only respond to commands appropriate for the types of peripherals serviced by it.

The object oriented processor 110 (which is described in detail below) has the task of routing messages to and from the object orient processor 112, as well as supplying its own functionality such as supporting hardware device(s) 134. The object oriented processor 112 may be the same as any one of the processors 104, 106, 108.

According to the present invention, an enhanced command language is provided wherein messages are prioritized and addressed and may be multi-cast to several object oriented processors.

According to a presently preferred embodiment of the invention, the high level command language includes three distinct message types: {command}, [response], and (exception), where the delimiting braces "{}", brackets "[]", and parenthesis "()" designate the type of message contained therebetween.

The presently preferred format of a command message can be either {toAddr fmAddr function parameters ~ checksum} or {start:end fmAddr function parameters ~ checksum}.

"toAddr" is an optional one byte (two hex characters) address of the intended recipient. If no recipient address is given, the command is a global command which is interpreted by all other processors in the system. According to a presently preferred embodiment, address codes may use digits 0 through 9 and lower case letters a through f and the address 00 is reserved for the host.

"start:end" represents a range of addresses which is optional in lieu of a single recipient address.

"fmAddr" is a two character mandatory sender address and, according to the presently preferred embodiment, 00 is the reserved address for the host processor.

"function" is a mandatory two character code. According to a presently preferred embodiment, the first character of a function is always an upper case letter and the second character may be an upper case letter or a digit 0-9. It will be appreciated that the change in character case between the address fields and the function field serves as an effective delimiter of these fields.

"parameter" is an optional field of arbitrary length depending on the function called. According to a presently preferred embodiment, any reserved characters used in the parameter field are literalized using the "\" character.

"~" indicates the start of an optional checksum.

An example of a global command is {00ZZ} by which the host sends a soft reset to all of the object oriented processors. An example of a command addressed to a range of addresses is {08:1400PSA00A02} which is sent from the host to object oriented processors at addresses 8 through 14 inclusive calling the function PS with the parameter string A00A02. The same command could be sent with a checksum in which case, the message would be {08:1400PSA00A02~C9}. The checksum value is calculated for all of the characters to the left of the tilde.

Response messages have the syntax [toAddr fmAddr data ~checksum] or [interpId fmAddr data ~checksum]. The first form of a response message is used when responding to a specific command and the second form is used when sending an unsolicited response, e.g. when sending data from an input device. The data field is optional and of arbitrary length. Any reserved characters used in the data field are literalized using the "\" character. The tilde indicates the start of a checksum. The starting field "interpId" indicates an unsolicited message and takes the form !s where s is a hex character indicating the internal source of the message within the sending processor. Unsolicited messages are interrupt driven as described in more detail below. An example of an unsolicited response message is [!503FE56~0A] where data FE56 is sent from source at address 5h in the processor having address 03h. As mentioned above, the address 00h is reserved for the host. An unsolicited response message may typically be generated by a processor which is coupled to an input device. For example, the processor 104 in Figure 1 would generate an unsolicited message in response to a keypress on the keyboard 120, or in response to one of the switches 122, 124 being switched. The internal source ID carried in the unsolicited message would indicate whether the data was generated at the keyboard 120, the switch 122, or the switch 124. It will be appreciated that the unsolicited messages do not contain a recipient address so that any processor interested in the message can interpret it. An example of a solicited response

message is [0003FE56] where the data FE56 is sent from the processor having address 03 to the host in response to some command from the host. For example, the processor 104 in Figure 1 would generate a solicited message in response to a keypress on the keyboard 120 following a command from the host to display a prompt on display 116 or 118 for a user to enter a keypress. As with the command messages, the checksum field is optional in response messages.

Exception messages have the syntax (toAddr fmAddr errNo ~checksum). Both address fields are mandatory. The errNo field is a hex coded error message and the checksum field is optional. According to the presently preferred embodiment, exception messages are not returned in response to global or multicast command messages in order to simplify flow control.

From the foregoing, it will be appreciated that any processor in the system 100 shown in Figure 1 can send a command to any other processor in the system and that the multicast form of messages allows a processor to send a single command to a group of processors. For example, in response to activation of the switch 122, the processor 104 could be programmed to send a command to the processor 108 to effect a change in the motor control 132; or data acquired at 130 could be sent by the processor 108 to the processor 104 for display on the display 118. In addition, when the system is designed, the designer may choose to allocate addresses of similar processors in a contiguous range so that a single command, to display a prompt, for example, is multicast to be interpreted by all the processors capable of displaying the prompt.

The choice of message delimiters and the interrupt flag is not arbitrary. When using a zero-dominant bus protocol, there is an inherent priority structure. The lower the ASCII value of the delimiter or flag, the higher its priority. Thus, commands, which are delimited by braces {}, have a lower priority than responses, which are delimited by brackets [], which in turn have

a lower priority than exceptions which are delimited by parentheses (). The interrupt flag ! has the highest priority.

According to the presently preferred embodiment, a plurality of predefined object oriented processors are provided for use by a developer. Each of the predefined processors has a specific functionality or functionality set and an address which is assigned at boot time. As discussed in my previous application, the presently preferred manner of constructing object oriented processors is by way of a virtual machine. That is, a general purpose processor is programmed to behave in a specific manner in response to commands from the high level language and to generate responses and exceptions in response to commands and in response to other events such as input events from a connected peripheral. Accordingly, the presently preferred object oriented processors are embodied as general purpose processors with a ROM program. However, from the discussion herein and in my previous applications, those skilled in the art will appreciate that the functionality of an object oriented processor can be achieved with any mixture of hardware and software. Generally, functionality which needs to be available in parallel with other functionality will require additional hardware; and functionality which can be available in series can be achieved with additional software. In addition, according to the invention, a developer may create unique object oriented processors which understand the high level command language of the invention.

As mentioned above, the addressable port in processor 110 is designed to allow for soft address allocation during operation of the system. That is, the "virtual address" of the port 110d in processor 110 can be changed by any of the other processors using a command function. The virtual address of the port 110d will then be the "effective address" of the processor 112. The link between the addressable port of a parent processor 110 and a child processor 112 utilizes a point to point link where addressing is not an issue. The virtual address of the child processor 112 is stored by the parent processor 110 in a database as described in more detail below. For example, assuming that

the processor 110 has the address "10", the addressable port 110d is assigned the address "12", and the processor 104 has the address "04", a message from the child processor 112 to the processor 106 will take the form {06PSA08A09} as it is received by the parent processor 110. The parent processor will insert the virtual from address of the child processor before sending the message to the bus in the form {0612PSA08A09}. Similarly, a message from the processor 106 to the processor 112 will take the form {1206PSA08A09}. Processor 106 will place the message on the bus and the processor 110 will recognize it as destined for the addressable port 110d. The processor 110 will then strip off the virtual address of the child processor 112 and forward the message in the form {06PSA08A09} to the processor 112. While the processor 110 has been shown with one addressable port, the invention contemplates providing a processor with several child ports with soft addressability.

According to a presently preferred embodiment of the invention, the command to set a virtual address takes the form {toAddr fmAddr ZPpqrs} where the parameter pq identifies which addressable port of the processor is to be assigned the soft address and the parameter rs is the soft address.

This soft addressing feature of the system taken together with the range addressing feature allows the system to regroup processors for multicast commands in response to changes in the system during operation as described in more detail below.

Turning now to Figures 1 and 2, according to the methods of the invention, functionally identical equivalents of the object oriented processors of the discrete system of Figure 1 are embodied on an ASIC chip to which the system is easily migrated after its full functionality is defined using the host processor. For example, as shown in Figure 2, an ASIC chip 200 includes the functional equivalent 204 of the processor 104 of Figure 1, the functional equivalent 206 of the processor 106, the functional equivalent 208 of the processor 108, the functional equivalent 210 of the processor 110, and the functional equivalent 212 of

the processor 112. Moreover, the functional equivalent processors are coupled to each other in a functionally equivalent manner with an on-chip bus 214 (which need only be as fast as the discrete bus) which is accessible off the chip via pins 214p shown schematically in Figure 2. To the extent that the functionally equivalent processors require connection to external peripherals, additional pins 204p, 206p, 208p, 210p, 212p are provided on the chip as shown schematically in Figure 2.

Thus, one method of developing an ASIC according to the invention includes selecting from a library of discrete object oriented processors to develop an implementation and selecting from a library of ASIC chips each of which embodies a plurality of object oriented processors. The developer designs and tests the system using discrete object oriented processors and then chooses an appropriate ASIC which embodies the object oriented processors necessary for the designed system. Since the ultimate functionality of the system is based on the high level command language, the functionality of the bus, and the choice of which kind of object oriented processors to use, the functionality of the ASIC system will be identical to the discrete component system. The ultimately selected ASIC may actually have more object oriented processors embodied on it than necessary for the application. Nevertheless, this method is cost effective because the ASICs are an off-the-shelf item.

According to another embodiment of the invention, the ASIC chip may also be provided with an on-chip host processor. This embodiment is shown in Figure 3 where the ASIC 300 is substantially the same as the ASIC 200 with the addition of an on-chip host processor 202. The ASIC 300 is also provided with a programming port 314p for programming the host processor for a specific application. Alternatively, the host processor may be pre-programmed with an on-chip ROM.

According to still another embodiment of the invention, a developer may design one or more custom object oriented processors for use in a discrete system and request custom ASIC

chips containing the object oriented processors designed for a particular application.

As mentioned above, a new object oriented processor according to the invention includes one or more child ports with soft addressability. In addition, the invention provides several other improvements to the object oriented processors of the parent applications.

Turning now to Figure 4, an object oriented processor 410 is shown which combines the functionality of several of the object oriented processors of Figure 1 including the object oriented processor 110 with child ports with soft addressability. The object oriented processor 410 includes a comms interface 410a which is embodied here as a multi-master RS-485 interface. Accordingly, those skilled in the art will appreciate that the presently preferred communication bus is an RS-485 bus. The object oriented processor 410 includes an enhanced intelligent message handler and flow controller generally indicated by 410b which includes a state driven parser and database 412, an RS-485 output queue manager 414, an output queue manager 416, 418 for each addressable port, an interrupt processor 419, an output gating function 420, 422, and 424 for each task specific functionality which is enabled to send messages and an output gating function 426, 428 for each addressable port. The processor 410 is also provided with a task specific functionality in the form of a hardware interface 410c and two child ports with soft addressability 410d, 410e.

The hardware interface 410c generally includes an LCD port 430 for outputting data to an LCD, a parallel output port 432, a user output port 434, a keyboard input port 436, a serial input port 438 for a magnetic card reader or an IR remote control receiver, and an analog input port 440 with A/D conversion and signal processing. Each of the input ports 436, 438, 440 is provided with a FIFO buffer 442, 444, 446 for holding received data until it is sent to the comms interface 410a. As shown in Figure 4, data from the FIFOs 442, 444, 446 is released for

transmission onto the bus by a corresponding output gating function 420, 422, 424.

Each of the child ports with soft addressability 410d, 410e generally includes a message receiver 448, 450 and an associated FIFO buffer 452, 454. As shown in Figure 4, messages in the FIFOs 452, 454 are released for transmission onto the bus by a corresponding output gating function 426, 428. The message receivers 448, 450 take messages from child processors coupled to the child ports with soft addressability 410d, 410e and add the virtual fmAddr to the messages based on information in the database 412.

As seen in Figure 4, all messages received from the bus by the processor 410 are first processed by the state driven parser and database 412. The database includes the assigned virtual addresses of the ports 410d, 410e as well as sufficient knowledge about the command language and the functionality of the processor to route incoming messages. The state driven parser and database 412, therefore, analyzes messages on the bus and determines whether the messages are intended for this processor 410 and where the messages should be routed within the processor 410. Thus, messages interpreted by the state driven parser and database 412 are appropriately forwarded to any one of the output ports 430, 432, 434, the child ports with soft addressability 410d, 410e, the interrupt processor 419, or the gating functions 420, 422, 424 of the input ports. When messages are forwarded to the child ports with soft addressability 410d, 410e, the messages are buffered by the output queue managers 416, 418.

Also as seen in Figure 4, all messages destined for output onto the bus from the processor 410 are released onto the bus by the output queue manager 414. The output queue manager 414 services messages according to priority before sending them onto the bus. The priority of messages is determined by the interrupt processor 419 which is configurable by the host processor. As shown in Figure 4, the interrupt processor 419 is coupled via an event bus to each of the input ports 436, 438, 440 and the

addressable port receivers 448, 450, and the interrupt processor 419 is thereby advised each time new data or messages enter the processor 410 from functional layer or the child processors. Similarly, the interrupt processor 419 is coupled to each of the output gating functions 420, 422, 424, 426, 428, and the interrupt processor 419 decides when data in the various FIFOs 442, 444, 446, 452, 454 will be released to the output queue manager 414.

According to the presently preferred embodiment, the interrupt processor assigns a priority value of from 0-255 to each message it identifies via the event bus. The output queue manager 414 utilizes these priority values to determine the active output stream for messages placed on the bus by the processor 410. The manner in which the interrupt processor assigns a priority value can be configured by a message from the host or another processor which is routed to the interrupt processor by the state driven parser and database 412. In addition, according to the preferred embodiment, the interrupt processor is provided with a timer so that messages may be sent automatically by the processor 410 according to a schedule which is configured by the host or another processor. The timer may also be used to limit the frequency of unsolicited message packets.

From the foregoing discussion of Figure 4, those skilled in the art will appreciate that the enhanced features of the processor 410 may also be applied to any of the object oriented processors described in the parent application and that these featured are not limited to use with a processor having the task specific functionality of the processor 410.

From all of the foregoing discussion, it will also be appreciated that the enhanced functionality of the object oriented processors described herein and the enhanced command language greatly relieves the host processor from much participation in the distributed processing system. Accordingly, the primary role of the host processor in the described

embodiments is to initialize and configure the object oriented processors and to resolve error messages which appear on the bus. Thus, it will be appreciated that the specific functionality of the system developed according to the invention is primarily determined by which kinds of object orient processors are selected for inclusion in the system. Once the types and number of object orient processors are selected and coupled to the communications bus, the host is used to configure the system by providing specific parameters for each of the object orient processors at the time of system initialization. Therefore, it will be understood that in some systems, the functions of the host may be provided by a relatively simple processor containing the initial configurations of each of the object orient processors.

An exemplary command language vocabulary for the processor 410 includes commands for configuring and controlling an LCD, a keypad, an encoder, a serial interface, and an A/D interface, as well as commands to effect system functions such as reset. The vocabulary of the processor is stored in the state driven parser and database and is preferably organized according to classes and methods where classes refers to the general functionality such as LCD display, keyboard scanning, etc. and methods refers to specific functionality within a class such as writing a string to an LCD display, etc. The exemplary processor 410, therefore includes a vocabulary of six classes, the methods of which are show in the following Tables 1-6.

COMMAND	FUNCTION
CAp	Select active LCD where p is the number of the active LCD (a line of an LCD display)
CDxpq	Direct byte to LCD where x is the state of the RS line and pq is the byte to be sent
CF<string>	Custom font
CIxpq	Initialize wher x is the LCD module, p is the number of lines, and q is the width
CLpq	Cursor location where is pq is the column of an active line
CMp	String Write Mode where p=
	0 = std line by line
	1 = scroll bottom right to top left
	2 = scroll top left to bottom right
CO	Display/Cursor/Blink On/Off
CS<string>	Write string
CX	Clear active LCD
CZ	Clear all displays

Table 1

Table 1 shows the commands for configuring and operating an LCD display which is attached to the LCD port 430 of the processor 410 shown in Figure 4. It will be appreciated that many of these commands will include arguments or parameters. When the state driven parser and database 412 sees a command beginning with the upper case character C, it is immediately determined that the command is meant for the LCD controller. The first character of the command is defined as the class of the command and the second character of the command is defined as the method within the class.

KMx	Define mode x = 0 4x4 keypad, no encoder x = 1 4x2 keypad, single encoder
KR	Read KB buffer
KTpq	Set KB debounce time where pq is the debounce time in milliseconds
KIpq	Set keyboard interrupt priority wher pq is the priority 0-255

Table 2

Table 2 shows the commands for configuring and reading a keypad which is attached to the keypad port 436 of the processor 410 shown in Figure 4. It will be appreciated that the keypad may be configured in several ways and that some of the keyboard resources may be exchanged for encoder resources if mode of the keyboard is set to 4x2. When the state driven parser and database 412 sees a command beginning with the upper case character K, it is immediately determined that the command is meant for the keyboard controller. In particular, the KR command is directed to the gate function 420 to release buffered keyboard data to the output queue manager. Other keyboard configuration commands are directed to the interrupt processor 419.

EIpq	Set encoder interrupt priority where pq is the priority 0-255
EPx	Read encoder position x = 0 read do not reset counter x = 1 read then reset counter
ER	Read encoder rate (detents per second)
ESx	Encoder saturation on/off

Table 3

Table 3 shows the commands for configuring and reading the optional encoder (when the keyboard is so configured). When the state driven parser and database 412 sees a command beginning with the upper case character E, it is immediately determined that the command is meant for the encoder.

SFpq	Define frame size (number of bits in a packet) where pq is the number of bits
SIpq	Set serial interface interrupt priority where pq is the priority 0-255

Table 4

Table 4 shows the commands for configuring the serial port 438. When the state driven parser and database 412 sees a command beginning with the upper case character S, it is immediately determined that the command is meant for the interrupt processor 419 to define the parameters of the serial port 438.

AIpq	Set A/D interrupt priority where pq is the priority 0-255
AMp	Set converter type where p is a type
ARpq	Read channel pq
AUpqrs	Set upper threshold for channel pq to be rs
ALpqrs	Set lower threshold for channel pq to be rs
AApq	Set the number of samples for signal averaging
ATpq	Set the measurement interval for signal averaging and threshold comparison

Table 5

Table 5 shows the commands for configuring and reading the A/D interface 440.

ZAx	Turn on/off acknowledge
ZCx	Turn on/off carriage return on packets
ZEx	Turn on/off error reporting
ZIx	Global interrupt enable/disable
ZPpqrs	Define objectLink child addresses pq = 00 define virtual address of Olink port A pq = 01 define virtual address of Olink port B rs assigned address
ZN	Get object's name
ZZ	Soft reset

Table 6

Table 6 shows the commands for system functions and in particular includes the command for functionality interrogation {ZN} and the command for defining child addresses. The {ZN} command can be used by the host to automatically configure and operate new functionality added to a distributed processing.

From the foregoing, those skilled in the art will appreciate that a basic distributed processing system according to the invention may be installed with a plurality of parent object oriented processors for providing functions such as display, keypad input, card swipe input, analog data acquisition, etc. Child processors may then be removably coupled to parent processors where the child objects provide additional functionality such as speech output at different locations in the system. As additional child processors are added to the system, the host finds the child processors by scanning the network with the {ZN} command which is issued for each possible address on the network. A simple change in the host software enables all of the functionality of the newly added child processor. While additional functionality can be added to the system by adding additional parent processors which also respond to the {ZN} command, there are several advantages to using child processors when expanding the functionality of the system. The child processors do not require the same sophisticated message handling system that the parent requires. Also, the child processors do not require the RS-485 multi-master bus interface. For these two reasons, the child processors can be much less expensive than the parent processors. In addition, local communication between child processors or between child and parent processors is removed from the system bus.

With the foregoing in mind, an example of a distributed processing system according to the invention is a building management system which operates security features, fire alarm features, and environmental control features. In such a system, distributed parent processors could be provided for each of several strategic locations. Child processors having speech messaging may be attached to selected parent processors and

addressed according to groups such as public areas, private areas, secure areas, etc. Environmental protection features can be easily added by providing child processors having data acquisition functions at appropriate locations. These child processors would provide signals relating to temperature, levels of pollutants, etc. The host processor could use this data to shut down or turn on various HVAC equipment, to signal voice messages at various locations in the system, etc.

As mentioned above, an object oriented processor according to the invention may be provided with a fully programmable functional layer wherein the exact functionality of the processor is determined by the developer. Such a processor would include a communications interface and intelligent message handling functions in the form of a shell such that the developer can define commands and message syntax to support the defined functionality of the programmable functional layer.

Referring now to Figures 5 and 6, an object oriented processor 510 according to another embodiment of the invention generally includes a shell layer 512 and an active object layer layer 514. The shell layer includes a communications interface 516, a global input parser 518, an output message former 520, an output message registrar 522, a scheduler 524, a system object 526, a memory manager 528, and an object (cell) library 530. The shell layer is generally provided in a readable memory and the active object layer is generally implemented in a writable memory. As described in more detail below, the system object 526 is automatically instantiated in the writable memory of the active object layer 514 at the time power is applied to the processor 510. The system object 526 is used in part to configure functions from the cell library 530. As shown in Figure 6, each cell, e.g. 540, in the cell library 530 has a functional layer 542 which includes code for performing a particular function, a parser layer 544 which includes code for interpreting messages, a timing layer 546 which includes code for scheduling, and an instantiation layer 548 which includes code

for self-instantiation of the cell 540 in the writable memory of the active object layer 514.

Referring now to Figure 7, the object oriented processor 510 according to the invention is preferably embodied on a single chip having a plurality of pins, e.g., p0 through p39. Two pins, e.g. p0 and p1 are preferably reserved for a communications link with a host processor 550 or other processor (not shown) via a network or point-to-point link 551; and two pins, e.g. p38 and p39 are preferably reserved for connection to a DC power source 552. According to the invention, the host processor 550 is used to configure the object oriented processor 510 utilizing a high level command language and may also be used to communicate with the object oriented processor during normal operations. As shown in Figure 7, additional object oriented processors 510a, 510b, 510c, e.g., may be coupled to the same bus 551 and may be configured with the same host 550. In addition, as mentioned above, the host 550 may be replaced by one or more boot ROMs (not shown). The processors 510, 510a, 510b, 510c may communicate with each other via the bus 551, with or without host intervention and one processor 510 may be used to configure another processor 510a, e.g.

Referring now to Figures 8 and 9, and with reference generally to Figures 5-7, when the object oriented processor 510 is powered on at 600 in Figure 5, the system object 526 automatically instantiates itself at 602 in Figure 9 in a portion 514a of RAM in the active object layer 514. During auto-instantiation, the system object 526 also reserves a portion of RAM 514b for maintaining an active task list, a list of pointers to cells in the cell library 530. The processor 510 is thus in a condition to receive high level commands from the host 550 which will assign specific functionality to one or more of the pins p2-p37 of the processor 510.

A typical configuration command from the host 550 to the object oriented processor 510 takes the form {zF(ENC4)}, where z is the address of the system object, F is the command to

instantiate, and ENC4 is the name of a cell in the cell library 530, i.e. a 4-wide encoder. According to a presently preferred embodiment, the names (addresses) of the different cells are given functionally, e.g. LCDT (text LCD controller), ENC4 (4-wide encoder), KB44 (4x4 keypad controller), etc. In response to this command, given at 604 in Figure 9, the global input parser 518 checks the syntax of the command at 606 in Figure 9 and passes the command to the system object 526. The system object sends a call at 608 to the instantiation layer 548 of the cell "ENC4" and tells the cell "ENC4" to instantiate itself. In response to the command from the system object, the instantiation layer of the cell "ENC4" checks at 610 its predefined area of reserved memory 514e for prior instantiations of "ENC4" and determines at 612 whether there are sufficient hardware resources available for an(other) instantiation. According to a presently preferred embodiment, each cell in the library is provided with a pre-coded address to a small block of RAM which is thus reserved for its use in keeping track of instantiations. If it is determined at 612 that insufficient resources are available, an error message is sent which is received by the output registrar at 614 and forwarded to the host which receives an error message at 616. If it is determined at 612 that sufficient resources exist for the instantiation, the instantiation layer of "ENC4" calls the memory manager 528 and requests at 618 an allocation of RAM sufficient for its needs. According to a presently preferred embodiment, the memory manager 528 maintains a pointer to the next byte of available memory in the "heap" as well as the address of the end of the heap. In response to a request for "n-bytes", the memory manager subtracts "n-bytes" from the end of heap address and compares the result to the heap pointer to determine at 620 whether there is enough RAM available. If sufficient RAM is not available, an error message is sent at 622 to the output registrar which passes the message to the host. If it is determined at 620 that RAM is available, the memory manager, at 624 in Figure 9, assigns the pointer to the instantiation of "ENC4" and increments the heap pointer by n-bytes. The instantiation layer of "ENC4" receives the pointer at 626 and writes the pointer at 628 to its block of reserved memory 514e.

As illustrated in Figure 8, the pointer points to the start of memory block 514c. According to a presently preferred embodiment, the cell "ENC4" allocates a portion of the RAM space assigned to it for output message headers and another portion of the RAM assigned to it for "private data". When the cell "ENC4" has been instantiated, the task dispatcher in the system object stores a pointer at 630 to the cell "ENC4" in the active task list. According to a presently preferred embodiment, the position in the active task list is used as the instantiation name of the instantiation of the cell. For example, if the active task list has six entries (a-f), the first instantiated cell will have the instantiation name "a", the second "b", the third "c", etc. Further communications with an instantiation of a cell will utilize this name.

After a cell has been instantiated as described above, pins can be assigned to it by the host using the command language according to the invention. For example, a command of the form {aP(B)} from the host to the processor 510 is directed to the instantiated cell having the name "a" and utilizes the command P to assign pins where the parameter B is the location of the pins assigned to "a". As shown in Figure 9, such a command is issued at 632. Upon receipt of such a message from the host 550, the global input parser 518 checks the message at 634 for correct syntax and will generate an error message to the host if the syntax is incorrect. Based upon the address "a", the input parser will look at 636 for "a" in the active task list 514b and direct the message to cell "ENC4". As mentioned above, according to the presently preferred embodiment, "a" would be the first pointer in the active task list, "b" would be the second, etc. According to the example given herein, the pointer at "a" points to the cell "ENC4" and the parser will therefore forward the message at 638 to the cell "ENC4". The cell "ENC4" receives the addressed message and scans its reserved memory area 514e at 640 in Figure 9 to find the pointer to the assigned workspace of the named instantiation of itself, e.g. 514c. The pin numbers are then stored at 642 by the cell "ENC4" in the private data area of instantiation "a". Once the pins have been assigned to the

instantiation "a" of the cell "ENC4", the instantiation is operational and the pins are functioning with a default interrupt priority of zero.

Each time a new cell is instantiated as described above, the timing of all tasks is recalculated. Thus, as shown in Figure 9, the system object stops all tasks at 644 and calls the timing layer of all instantiations of cells. The instantiations respond at 646 with a worst case time to the system object and the worst case time is used to calculate the offset for the next active task. Each instantiation of a cell stores at 648 its offset in the private data area of its assigned RAM. The time between the worst case and the actual time used by each cell instantiation is used by the system object for background tasks. After the timing of all the tasks is recalculated, the system object returns at 650 to scanning the active task list which is described in more detail below with reference to Figure 10. As shown in Figure 9, timing may be recalculated in response to a host command at 652 to assign a particular priority level to a particular cell instantiation. The scheduling of priorities may is performed by the scheduler 524 which may be considered a part of the system object or a separate entity.

Turning now to Figure 10, the task dispatcher of the system object continually scans the active task list starting at 700 and periodically checks at 702 whether a new cell instantiation has been added. In actual practice, the scheduler sets timers for tasks based upon their priority and background tasks are completed when extra time is available. Therefore, the order of operations shown in Figure 10 is not necessarily the order in which operations will be scheduled by the scheduler. If a new cell instantiation has been added, the procedure described above (644-650 in Figure 9) is performed at 704 in Figure 10 and the system object then returns to scanning active tasks. This includes monitoring the output message former and the global input parser to determine whether messages need to be delivered. For example, at 706 it is determined whether an incoming message is pending (in the buffer of the global input parser). If so,

the input parser examines the active task list at 708 to determine the cell for which the message is addressed and passes the message to the cell. The parser layer of the cell examines, at 710, its preassigned reserved memory to determine which instantiation of itself should receive the message and passes the message to the appropriate layer (functional, timing, or instantiation) of the cell for processing as an active task. The system object then checks at 712 whether an outgoing message is in the queue of the output registrar. If there are messages in the queue, the output message former reads the highest priority pointer in the output registrar at 714. The pointer in the queue points to the output message header of the cell generating the message. As mentioned above, the output message headers contain pointers to output buffers in RAM as well as an indication of the type of data to be sent, and a flag to indicate whether the output message header has been queued, etc. At 716, the output message former uses the output message header and the data to create a message for output onto the network, sends the message, and then drops the queue flag. An example of a message format is {ToArraytoobjMethod(data)FromObjfromarray} where the fields are delimited by case, ToArray is the processor address of the recipient, toobj is the cell address of the recipient, Method is a function, data is data, and the last from fields indicate the address of the sender. If the from addressing is blank, the host is the sender. The system object then resumes scanning the active task list at 700.

According to other preferred aspects of the invention, a high level language is provided for communication between object oriented processors and a host processor during set-up and during operation. The language is also used by processors to communicate with each other.

The functionality of each cell in the cell library may be chosen from a broad range of functions such as those discussed in the parent applications. Object oriented processors according to the invention may be provided with a set of complementary

functionalities or may be provided with a broad range of different functionalities.

There have been described and illustrated herein several embodiments of methods and apparatus for distributed processing utilizing object oriented processors and rapid ASIC development. While particular embodiments of the invention have been described, it is not intended that the invention be limited thereto, as it is intended that the invention be as broad in scope as the art will allow and that the specification be read likewise. Thus, while particular communications buses have been disclosed, it will be appreciated that other buses could be utilized. Also, while specific exemplary high level command language messages have been shown, it will be recognized that the command language may be changed and/or expanded to accommodate different kinds of task specific functionality and different types of configuration variables. Moreover, while particular configurations have been disclosed in reference to the number and types of object oriented processors contained on a single ASIC chip, it will be appreciated that other configurations could be used as well. It will therefore be appreciated by those skilled in the art that yet other modifications could be made to the provided invention without deviating from its spirit and scope as so claimed.

Claims:

1. A method for rapid development of ASIC chips utilizing distributed processing architecture comprising:

a) designing a system of discrete components including a first host processor and at least one object oriented processor;

b) providing a high level command language for communication between the first host processor and the at least one object oriented processor;

c) coupling the first host processor to the at least one object oriented processor with a communications bus;

d) programming the host processor with the high level command language to define a functionality of the system embodied as a first program;

e) testing and debugging the system of discrete components;

f) replicating the functionality of the at least one object oriented processor on an ASIC chip having an on-chip bus to which the replicated functionality is coupled;

g) coupling one of the first host processor and a second host processor to the on-chip bus; and

h) programming the host processor coupled to the on-chip bus with the first program.

2. A method according to claim 1, wherein:

the at least one object oriented processor includes a plurality of object oriented processors.

3. A method according to claim 2, further comprising:

i) assigning each of the plurality of object oriented processors a unique address via a high level command language message.

4. A method according to claim 3, wherein:

said high level command language includes an addressing scheme for directing messages to a specific one of the plurality of object oriented processors.

5. A method according to claim 3, wherein:

said high level command language includes an addressing scheme for directing messages to a specific sub-group of the plurality of object oriented processors.

6. A method according to claim 1, wherein:

said step of designing a system of discrete components includes at least one child object oriented processor coupled to one of the at least one object orient processors, and

said step of replicating the functionality of the at least one object oriented processor on an ASIC chip includes replicating the functionality of the at least one child object oriented processor on the ASIC chip and coupled to the replicated functionality of the one of the at least one object orient processors on the ASIC chip.

7. A distributed processing system, comprising:

a) a host processor;

b) a communications bus coupled to said host processor;

c) a plurality of object oriented processors coupled to said communications bus; and

d) a high level command language by which said host processor and said plurality of object oriented processors communicate, said command language including an addressing scheme by which messages from one of said host processor and said plurality of object oriented processors are directed to another of said host processor and said plurality of object oriented processors.

8. A distributed processing system according to claim 7, wherein:

said high level command language includes means for assigning an address to at least one of said plurality of object oriented processors.

9. A distributed processing system according to claim 7, wherein:

said addressing scheme includes means by which messages from one of said host processor and said plurality of object oriented processors are directed to a subgroup of said host processor and said plurality of object oriented processors.

10. A distributed processing system according to claim 7, further comprising:

e) a child object oriented processor, wherein

one of said plurality of object oriented processors has port means for coupling to said child object oriented processor, and said child object oriented processor is coupled to said one of said plurality of object oriented processors via said port means.

11. A distributed processing system according to claim 10, wherein:

said port means has an assignable address whereby messages directed to an assigned address are received by said child object oriented processor.

12. A distributed processing system according to claim 7, wherein:

said high level command language includes message delimiters which define the priority of messages.

13. A distributed processing system according to claim 12, wherein:

said high level command language includes message delimiters for command messages, response messages, and exception messages.

14. A distributed processing system according to claim 13, wherein:

said high level command language includes an interrupt character for identifying unsolicited responses.

15. A distributed processing system according to claim 14, wherein:

unsolicited responses have priority over exception messages, exception messages have priority over response messages, and response messages have priority over command messages.

16. An object oriented processor for use in a distributed processing system, said processor comprising:

a) a communications interface;

b) an intelligent message handler; and

c) a functionality layer, wherein

said intelligent message handler includes internal flow control means for controlling the flow of messages to and from the functionality layer, said internal flow control means being externally configurable.

17. An object oriented processor according to claim 16, wherein:

said intelligent message handler includes a state driven parser and database, said database including message information relating to said functionality layer.

18. An object oriented processor according to claim 17, further comprising:

d) port means for coupling said object oriented processor to a child processor, wherein

said object oriented processor has a communications address and said port means has an assignable communications address.

19. A configurable object oriented processor for use with a source of instructions, said object oriented processor comprising:

a) a readable memory containing a plurality of configurable functions and a system function;

b) communications interface means for receiving instructions from the source of instructions;

c) intelligent message parser means for reading instructions; and

d) writable memory means for storing a configured one of said configurable functions, wherein

in response to an instantiation instruction, said message parser forwards the instantiation instruction to said system function and said system function instantiates a copy of one of said plurality of configurable functions in said writable memory,

in response to a configuration instruction, said message parser forwards the configuration instruction to said copy of one of said plurality of configurable functions and said copy of one of said plurality of configurable functions is configured.

20. A configurable object oriented processor according to claim 19, further comprising:

e) a plurality of input/output connections, wherein

in response to the configuration instruction, said copy of one of said plurality of configurable functions is configured by associating said copy of one of said plurality of configurable functions with a subset of said plurality of input/output connections.

21. A configurable object oriented processor according to claim 19, wherein:

a copy of said system function is automatically instantiated in said writable memory when said processor is powered-up.

22. A configurable object oriented processor according to claim 19, wherein:

the source of instructions is a user-operable host processor.

23. A configurable object oriented processor according to claim 19, wherein:

the source of instructions is a boot ROM.

24. A configurable object oriented processor according to claim 19, wherein:

in response to a plurality of instantiation instructions, said system function instantiates copies of a plurality of said plurality of configurable functions in said writable memory.

25. A configurable object oriented processor according to claim 24, wherein:

in response to the configuration instruction, one of said copies of one of said plurality of configurable functions is assigned an interrupt priority.

26. A configurable object oriented processor according to claim 25, further comprising:

e) output message handler means for handling a plurality of output messages from said copies of a plurality of said plurality of configurable functions, wherein

said communications interface means includes means for transmitting said output messages to a receiver.

27. A configurable object oriented processor according to claim 26, wherein:

the source of instructions is a host processor and the receiver is the host processor.

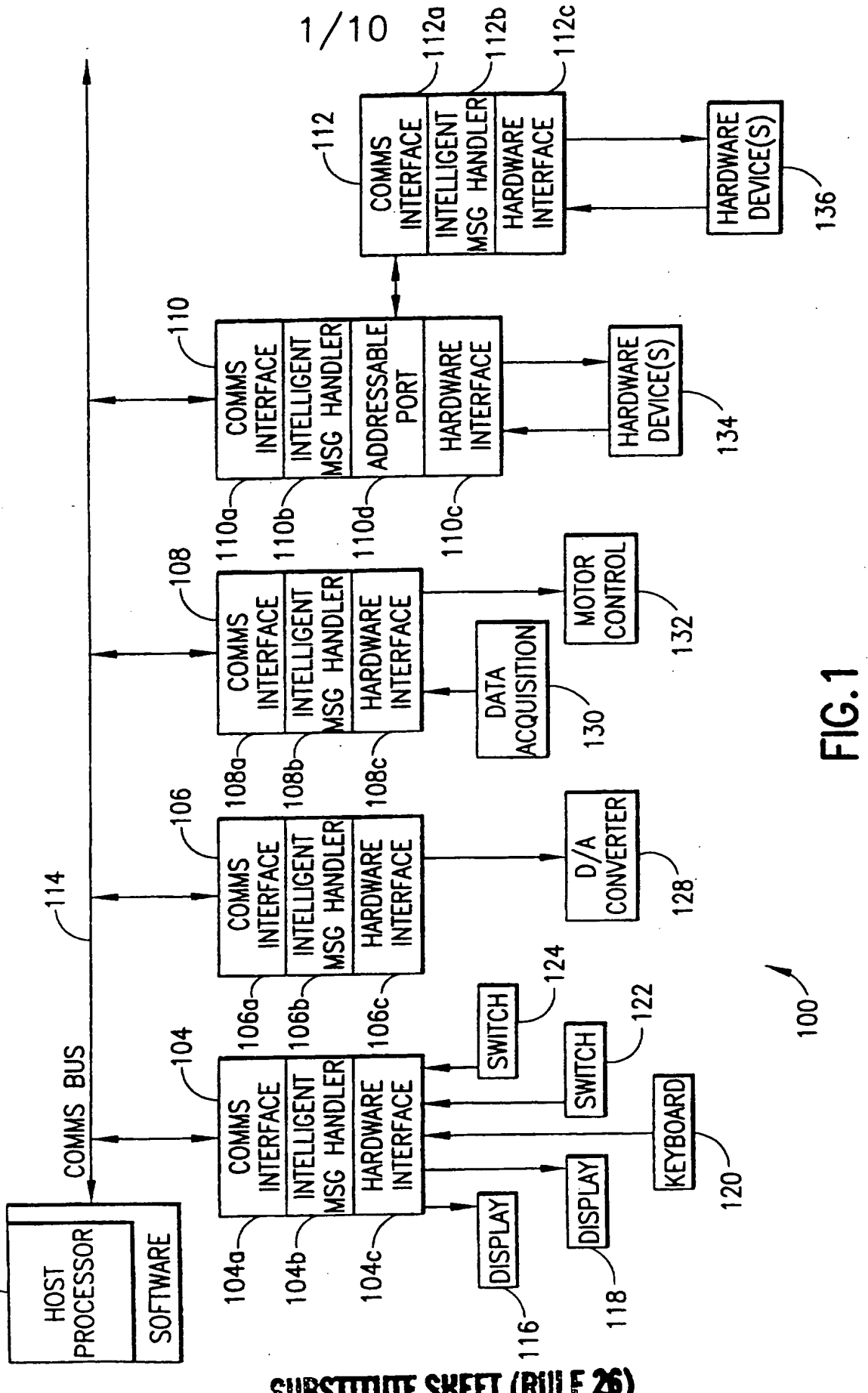

HOST PROCESSOR
SOFTWARE
102
COMMS BUS
114
COMMS INTERFACE
INTELLIGENT MSG HANDLER
HARDWARE INTERFACE
104
104a
104b
104c
106
106a
106b
106c
108
108a
108b
108c
110
110a
110b
110c
110d
ADDRESSABLE PORT
112
112a
112b
112c
DISPLAY
116
DISPLAY
118
KEYBOARD
120
SWITCH
122
SWITCH
124
D/A CONVERTER
128
DATA ACQUISITION
130
MOTOR CONTROL
132
HARDWARE DEVICE(S)
134
HARDWARE DEVICE(S)
136
100

FIG. 1

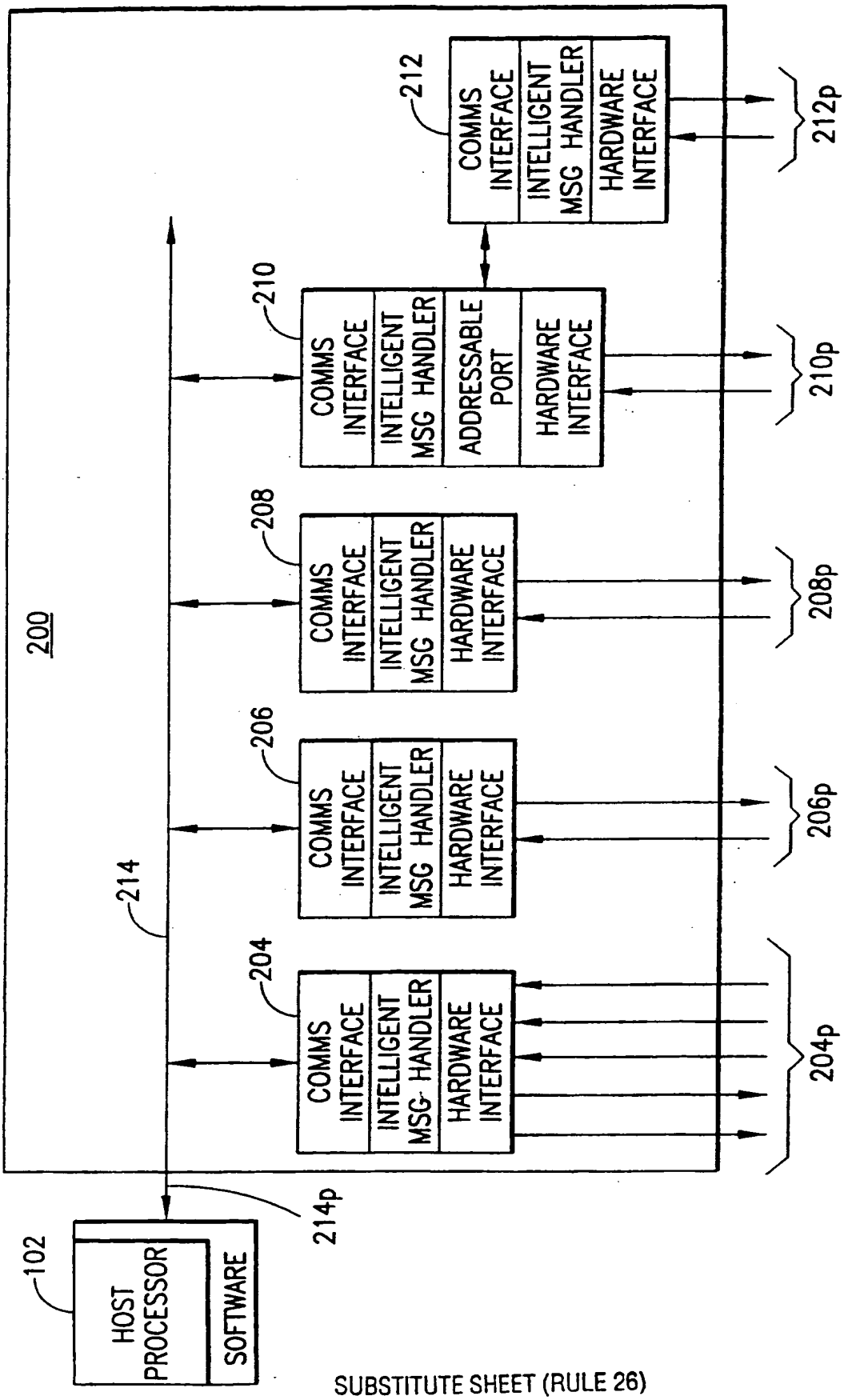

200
212
COMMS INTERFACE
INTELLIGENT MSG HANDLER
HARDWARE INTERFACE
212p
210
COMMS INTERFACE
INTELLIGENT MSG HANDLER
ADDRESSABLE PORT
HARDWARE INTERFACE
210p
208
COMMS INTERFACE
INTELLIGENT MSG HANDLER
HARDWARE INTERFACE
208p
206
COMMS INTERFACE
INTELLIGENT MSG HANDLER
HARDWARE INTERFACE
206p
204
COMMS INTERFACE
INTELLIGENT MSG- HANDLER
HARDWARE INTERFACE
204p
214
214p
102
HOST PROCESSOR
SOFTWARE

FIG.2

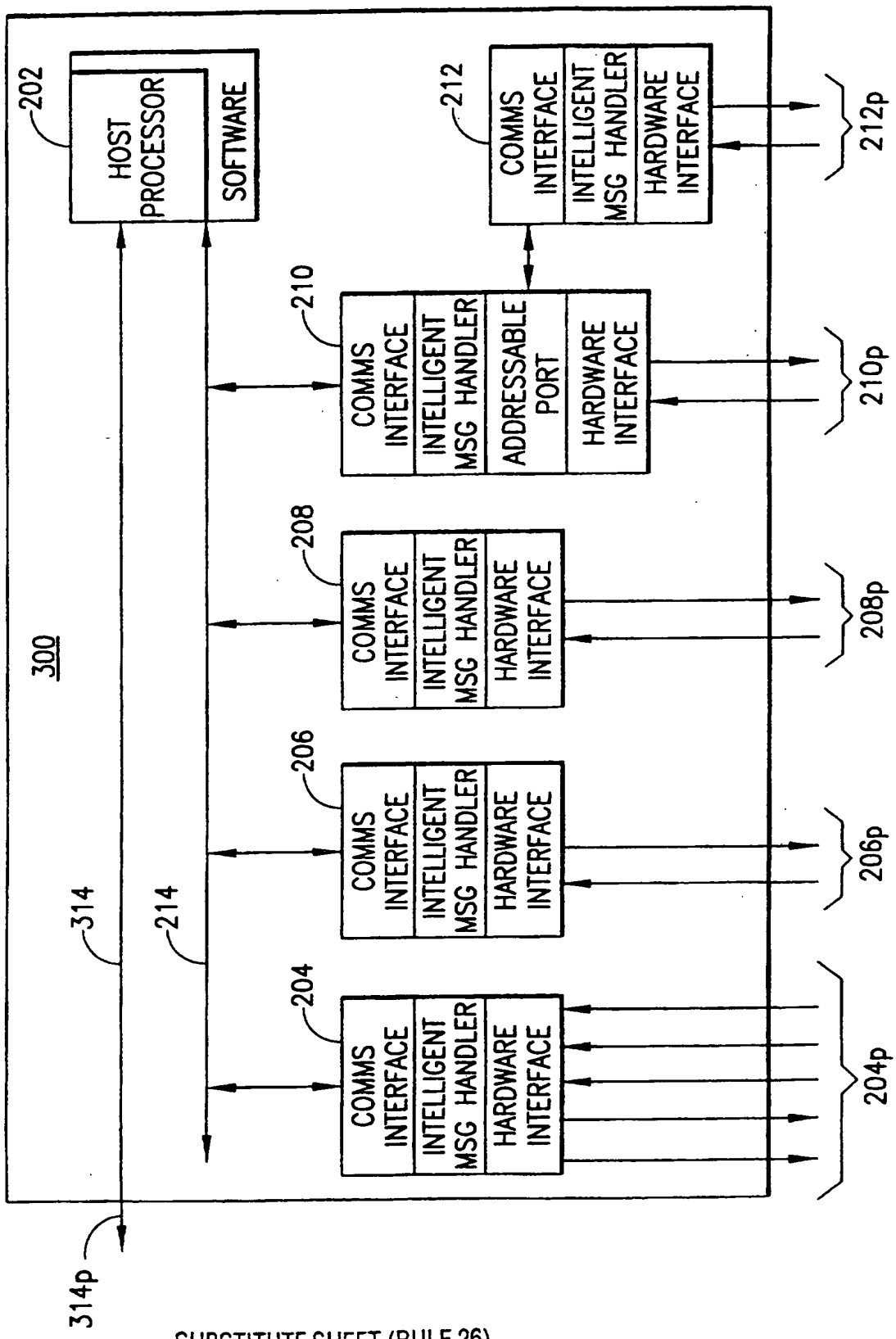

202
HOST PROCESSOR
SOFTWARE
212
COMMS INTERFACE
INTELLIGENT MSG HANDLER
HARDWARE INTERFACE
212p
210
COMMS INTERFACE
INTELLIGENT MSG HANDLER
ADDRESSABLE PORT
HARDWARE INTERFACE
210p
208
COMMS INTERFACE
INTELLIGENT MSG HANDLER
HARDWARE INTERFACE
208p
206
COMMS INTERFACE
INTELLIGENT MSG HANDLER
HARDWARE INTERFACE
206p
204
COMMS INTERFACE
INTELLIGENT MSG HANDLER
HARDWARE INTERFACE
204p
300
314
214
314p

FIG.3

FIG.4A
FIG.4B
FIG.4

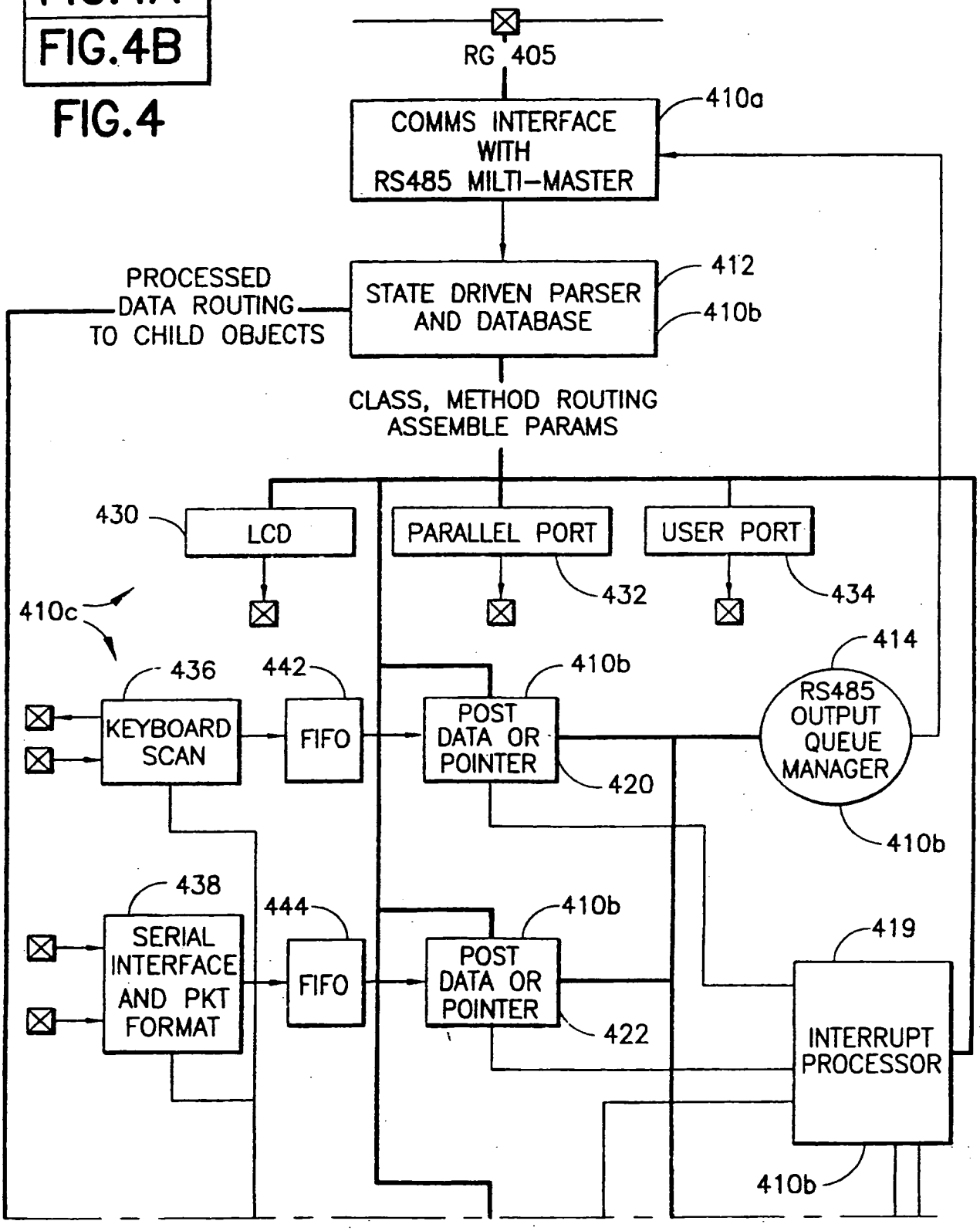


FIG.4A

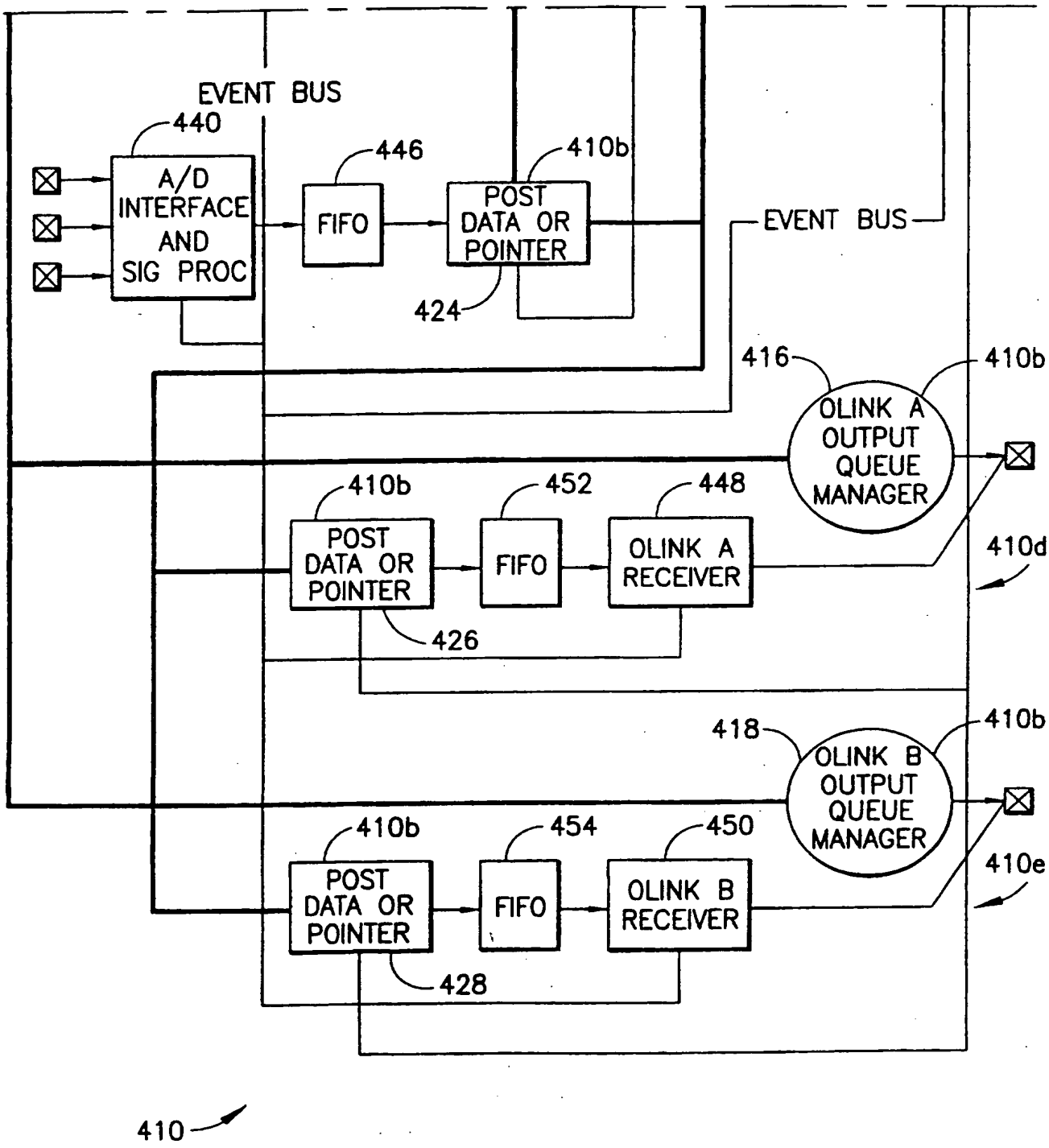

EVENT BUS
440
A/D INTERFACE AND SIG PROC
446
FIFO
410b
POST DATA OR POINTER
424
EVENT BUS
416
410b
OLINK A OUTPUT QUEUE MANAGER
410b
452
448
POST DATA OR POINTER
FIFO
OLINK A RECEIVER
410d
426
418
410b
OLINK B OUTPUT QUEUE MANAGER
410b
454
450
POST DATA OR POINTER
FIFO
OLINK B RECEIVER
410e
428
410

FIG.4B

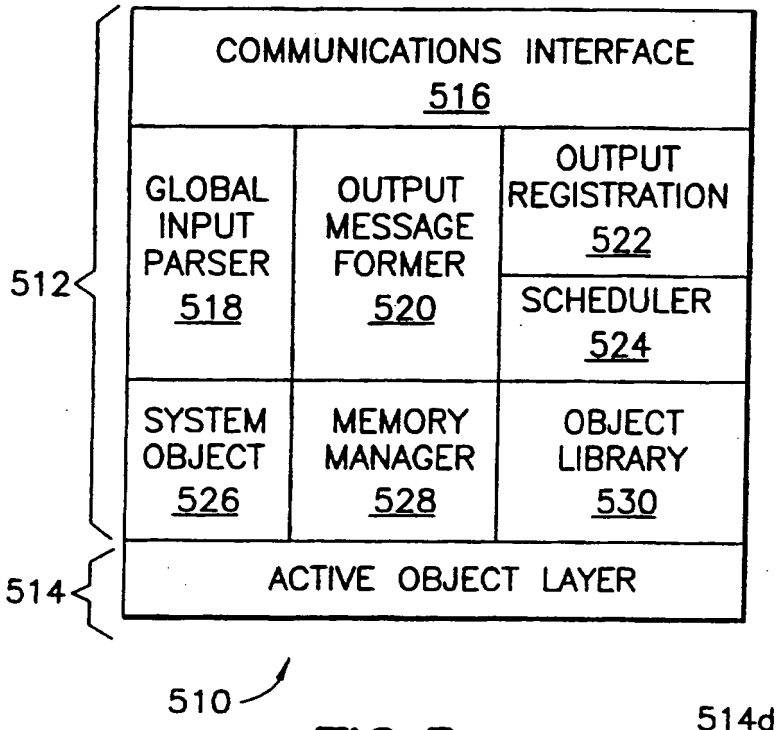


FIG.5

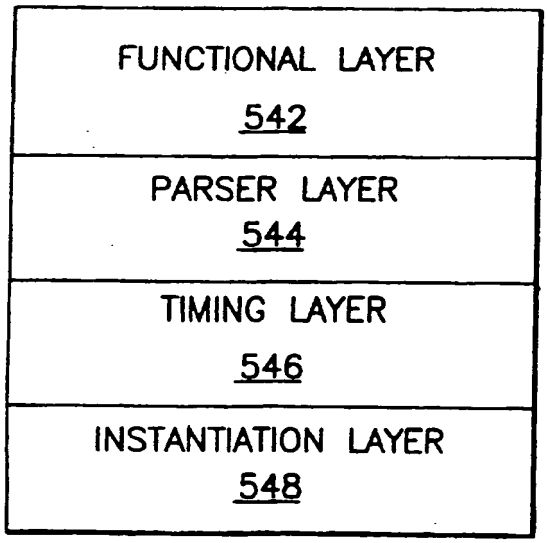


FIG.6

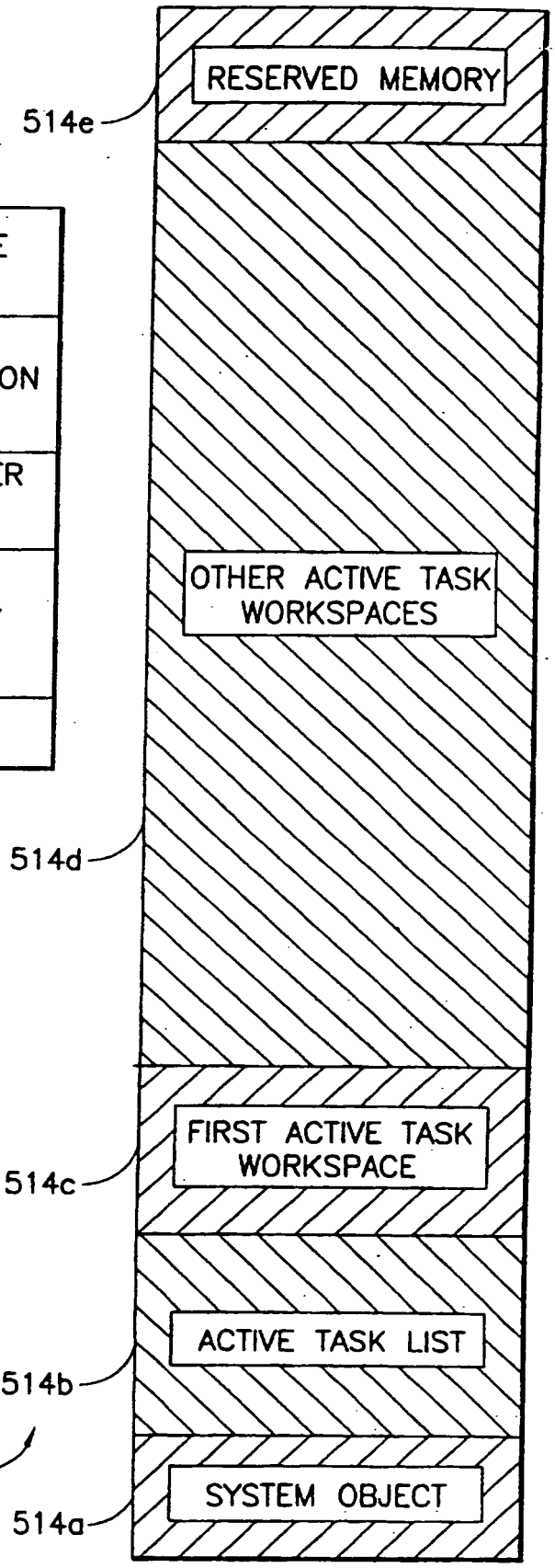


FIG.8

FIG.7

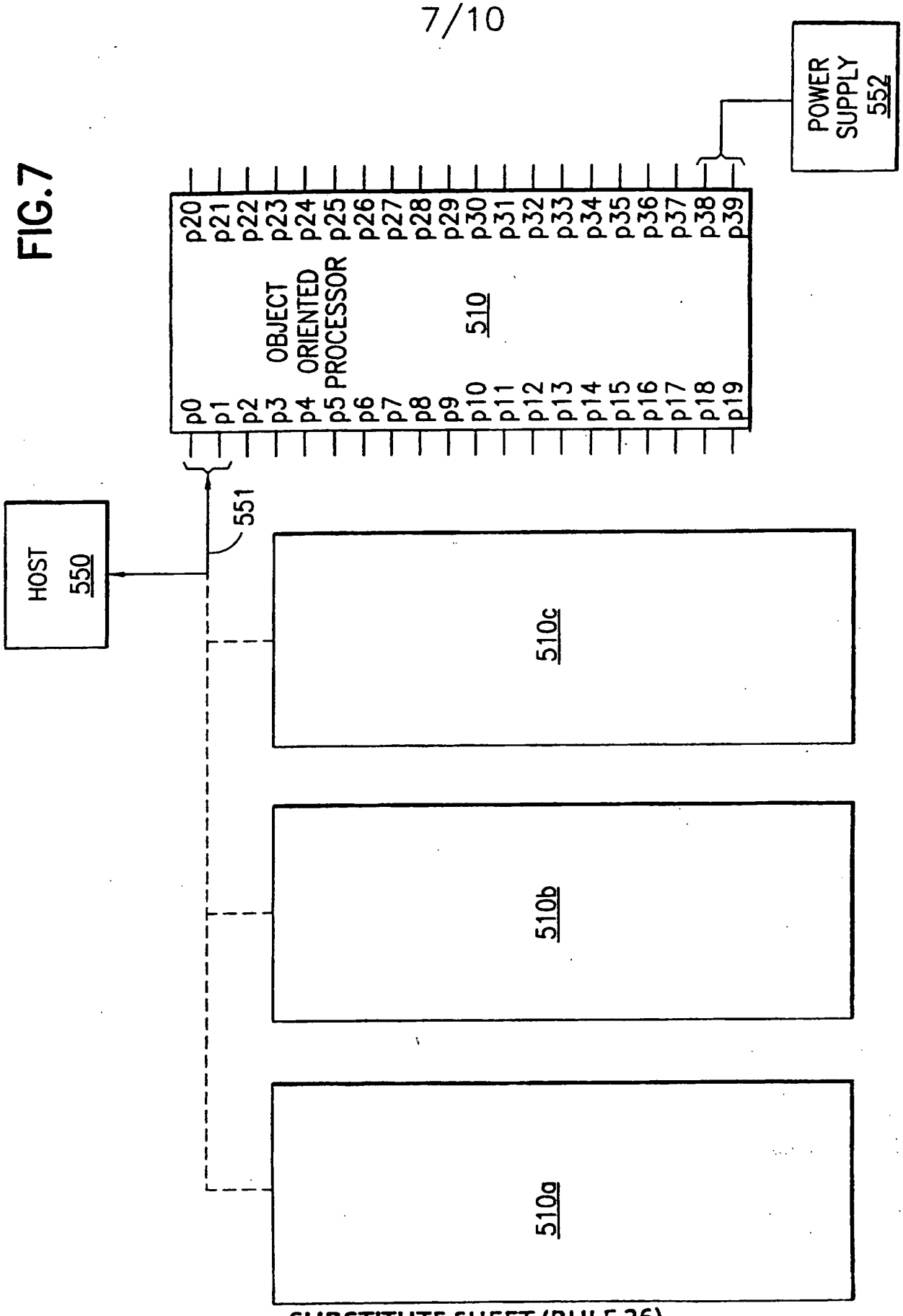

POWER SUPPLY 552
p20 p21 p22 p23 p24 p25 p26 p27 p28 p29 p30 p31 p32 p33 p34 p35 p36 p37 p38 p39
OBJECT ORIENTED PROCESSOR
510
p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12 p13 p14 p15 p16 p17 p18 p19
551
HOST 550
510c
510b
510a

FIG.9A
FIG.9B
FIG.9

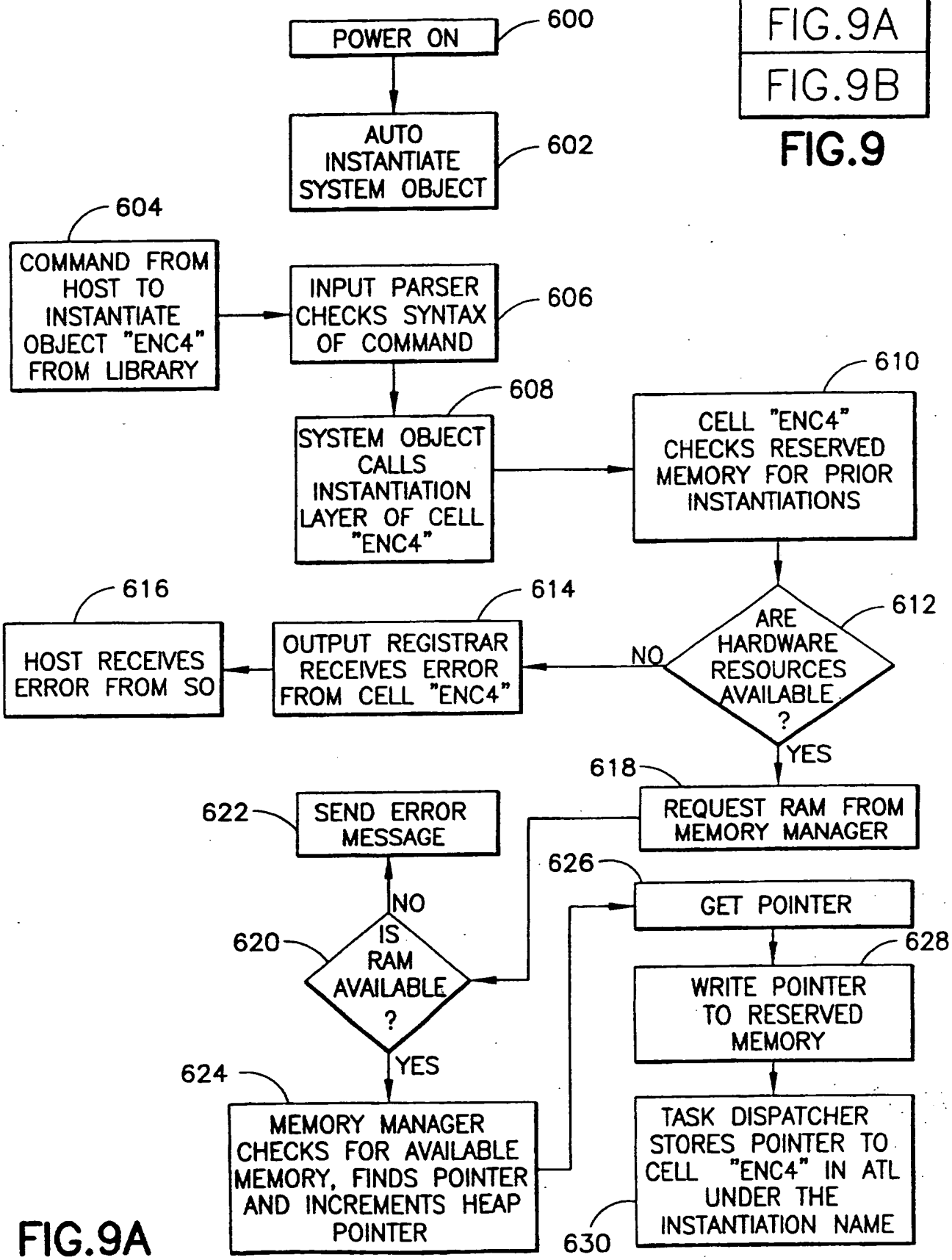


FIG.9A

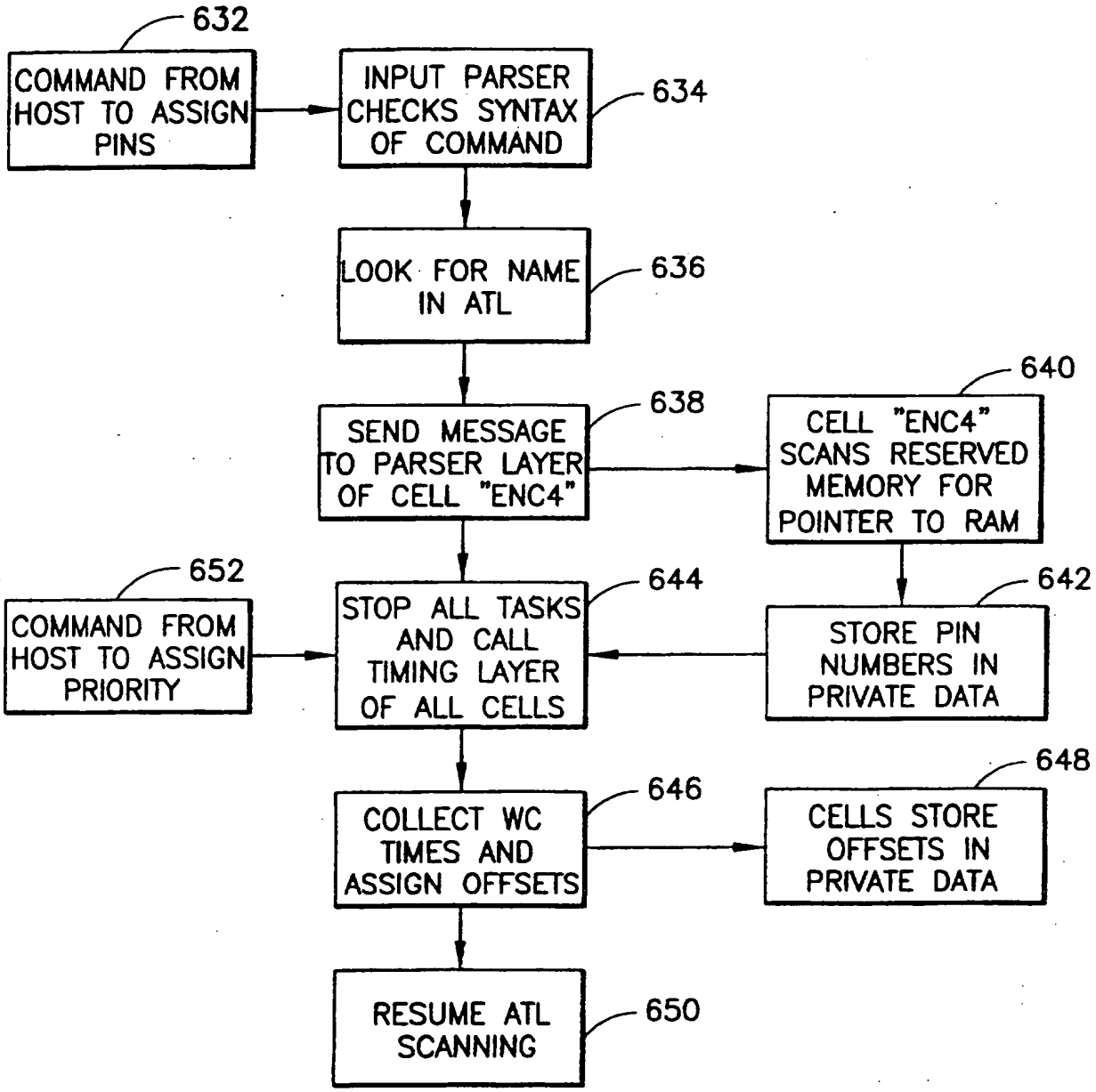

632
COMMAND FROM HOST TO ASSIGN PINS
INPUT PARSER CHECKS SYNTAX OF COMMAND
634
LOOK FOR NAME IN ATL
636
SEND MESSAGE TO PARSER LAYER OF CELL "ENC4"
638
CELL "ENC4" SCANS RESERVED MEMORY FOR POINTER TO RAM
640
652
COMMAND FROM HOST TO ASSIGN PRIORITY
STOP ALL TASKS AND CALL TIMING LAYER OF ALL CELLS
644
STORE PIN NUMBERS IN PRIVATE DATA
642
COLLECT WC TIMES AND ASSIGN OFFSETS
646
CELLS STORE OFFSETS IN PRIVATE DATA
648
RESUME ATL SCANNING
650

FIG.9B

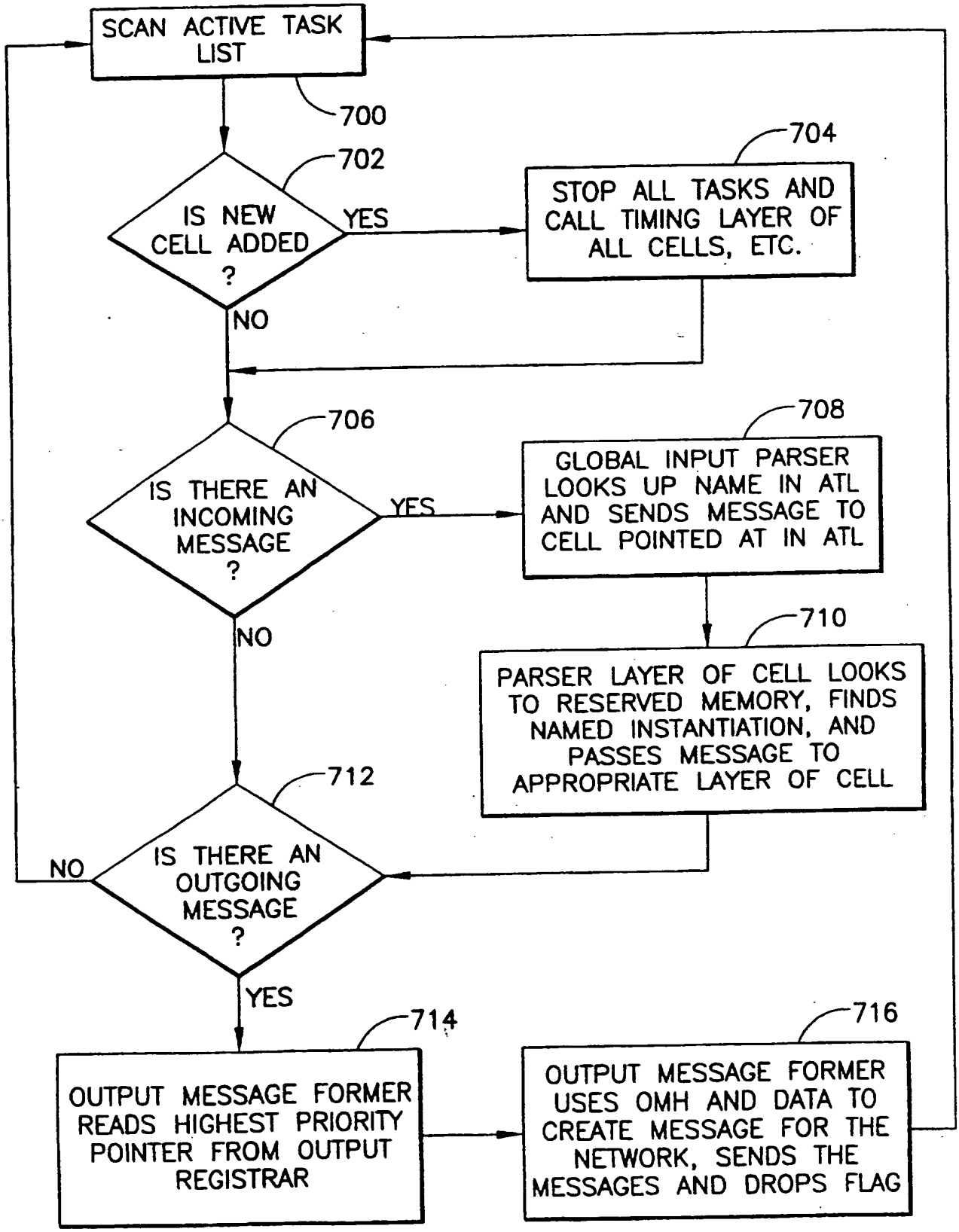

SCAN ACTIVE TASK LIST
700
IS NEW CELL ADDED ?
702
YES
704
STOP ALL TASKS AND CALL TIMING LAYER OF ALL CELLS, ETC.
NO
IS THERE AN INCOMING MESSAGE ?
706
YES
708
GLOBAL INPUT PARSER LOOKS UP NAME IN ATL AND SENDS MESSAGE TO CELL POINTED AT IN ATL
NO
710
PARSER LAYER OF CELL LOOKS TO RESERVED MEMORY, FINDS NAMED INSTANTIATION, AND PASSES MESSAGE TO APPROPRIATE LAYER OF CELL
IS THERE AN OUTGOING MESSAGE ?
712
NO
YES
OUTPUT MESSAGE FORMER READS HIGHEST PRIORITY POINTER FROM OUTPUT REGISTRAR
714
OUTPUT MESSAGE FORMER USES OMH AND DATA TO CREATE MESSAGE FOR THE NETWORK, SENDS THE MESSAGES AND DROPS FLAG
716

FIG.10

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/12516

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 15/00
US CL :395/800.28

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/800.28

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

NONE

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X,P	US 5,634,070 A (ROBINSON) 27 May 1997, cols. 7-21.	1-26
X	US 5,095,522 A (FUJITA et al.) 10 March 1992, Fig. 1 and col. 3, line 49 - col. 7, line 40.	1-26

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"B"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 07 OCTOBER 1997	Date of mailing of the international search report 02 DEC 1997
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer JOHN HARRITY Telephone No. (703) 305-9596

Form PCT/ISA/210 (second sheet)(July 1992)*